10. ročník Žilina 2. – 4. 7. 2018
Otvorený softvér vo vzdelávaní, výskume a v IT riešeniach
osscconf.soit.sk
OSS Conf
OSSConf 2024: 1–10

PF2020! aneb Když nestačí ani R, ani TeX

PAVEL STŘÍŽ (CZ)

Abstrakt. Článek zmiňuje úvahy a kroky vedoucí k vysázení loga PF2020! Logo je vysázené z šestiúhelníkových nálepek, více o kolekcích v GitHub repozitářích hex-stickers (Rstudio) a BiocStickers (Bioconductor). Hlavní zdroj inspirace bylo logo z konference useR! 2018, které vytvořil Mitchell O'Hara-Wild za použití jeho R skriptu hexwall.

Klíčová slova. R, ImageMagick, hexwall, raster, gglogo, magick, tidyverse, ggplot2, sf, TeX, Lua, Bash, GraphicsMagick, png, pacman.

PF2020! OR WHEN NEITHER R NOR TeX ARE SUFFICIENT

Abstract. The article describes a thinking process and a problem-solving approach of creating a PF2020! logo. It's typeset of hexagon stickers, see Rstudio's hex-stickers and Bioconductor's BiocStickers repositories in GitHub. The main inspiration came from the useR! 2018 logo which was created by Mitchell O'Hara-Wild using his hexwall R script.

Keywords. R, ImageMagick, hexwall, raster, gglogo, magick, tidyverse, ggplot2, sf, TeX, Lua, Bash, GraphicsMagick, png, pacman.

1. Problém typu word cloud

Word cloud / wordle / tag cloud se běžně řeší tak, že slova či obrázky se převedou do rastrových obrázků a následně se zkoumají přesahy na úrovni pixelů[1]. Dlouhodobě se zabýváme vektorovým řešením tohoto problému na úrovni TeXu. To lze zrealizovat např. přes METAPOST a příkaz `bisect`, ale to je vše „na dlouhé lokte". Proto nás zaujalo logo z konference userR! 2018, které má svou mřížku z šestiúhelníků, ale nálepky (angl. stickers) jsou sázeny jen uvnitř polygonů, v jejich případě mapy Austrálie. Návod lze nalézt na blogu autora, viz webová stránka `https://www.mitchelloharawild.com/blog/user-2018-feature-wall/`.

2. První dojmy

Zkusili jsme v tomto duchu připravit PF2020!, v R, co nejrychleji… Ale! Shrneme-li problémy: je potřeba doinstalovat knihovny v R a k tomu je potřeba mít nástroje a knihovny. Užili jsme Linux a omlouváme se uživatelům Microsoft Windows, že jsme testy nezkoušeli pod tímto operačním systémem. Instalovali jsme jednotlivé knihovny jednu po druhé a sledovali v R chybové zprávy. Tímto způsobem jsme

[1] `https://www.jasondavies.com/wordcloud/about/`

se vrátili pod Linux a doinstalovali vždy to, co bylo potřeba. A opětovně instalovali konkrétní knihovnu v R. Jednalo se postupně o knihovny `libmagick++-dev`, `librsvg2-dev`, `libssl-dev`, `libogdi3.2-dev`, `libcurl4-openssl-dev`, `gdal-bin` a `libwebp-dev`. Je však možné, že jsme některé nástroje již měli nainstalované, proto není výčet úplný.

Funkce `hexwall` je závislá na knihovně `magick`, který nám pro velký počet souborů přestane fungovat. V našem případě kolem tisícího souboru. Nehledě na časovou náročnost práce s obřím rastrovým obrázkem v pozadí.

Další problém byl, že se nám za půlden testů zaplnil pracovní adresář `/tmp/` o 500 GB. Hledejme proto jiné nástroje.

Poslední problém byl, že jsme nechtěli mapu (geografická data), ale texty, případně užít obecný (černobílý) rastrový obrázek. Na tohle jsme se zaměřili.

3. Zdárné kroky s knihovnou gglogo

Při hledání převodu znaků do polygonů jsme objevili tuto knihovnu. Pracovali jsme v adresáři, kam jsme si uložili skript hexwall:

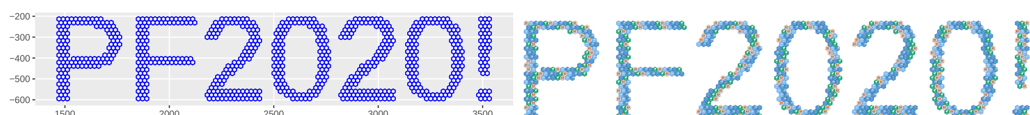
```
$ git clone https://github.com/mitchelloharawild/hexwall.git; \
> cd hexwall
```

Ačkoliv jsme zápasili s převody mezi datovými typy, zde je použitelný výsledek. Mezi pokusy jsme na vyčištění používali příkaz `rm(list=ls())`.

```
library(gglogo); library(raster); library(sf)
letter <- letterToPolygon("PF2020!", fontfamily="Helvetica", dim=c(5000,800))
Sr1 = Polygon(cbind(letter$x,letter$y)); Srs1=Polygons(list(Sr1),"s1")
SpP = SpatialPolygons(list(Srs1), 1:1)
hex_points <- SpP %>% spsample(type = "hexagonal", cellsize = 20)
hex_points@coords

library(ggplot2); library(tidyverse); as_tibble(hex_points@coords)
aus_hex <- HexPoints2SpatialPolygons(hex_points, dx = 20)
#pdf("nahled.pdf")
ggplot() + geom_sf(data=st_as_sf(aus_hex), colour="blue", fill=NA)
#dev.off()

source("hexwall.R")
mojkovo <- hexwall("samplehex", sticker_width=20, coords=hex_points@coords,
    sort_mode="filename")
plot(mojkovo)
#image_write(mojkovo,"pf2020-pres-hexwall.png")
```



V této ukázce se nám nelíbilo, že je tam málo nálepek (5 souborů), nemůžeme užít zúženou mezeru za PF i netradičně před vykřičník, abychom poškádlili typografický svět, a chtěli jsme si zkusit výběr bez vracení s vracením (vysvětlíme). Pokusme se v další části článku tyto úkoly vyřešit.

4. První krok s knihovnou raster

Jistým vývojovým mezistupněm se nám stala „zakázka“, chceme-li experiment, pro organizátory konference OSSConf v Žilině, `http://ossconf.soit.sk/`. Vzali jsme logo roku 2019, zvětšili, v GIMPu zasáhli do roku, drobně jsme roztáhli cifry a vyčistili vyhlazování typické u obrázků na internetu (Colors→Threshold) plus úprava pixelů „zde, tu a támhle“. Tím jsme si zajistili obrázek přesně se čtyřmi barvami a bílým pozadím (v případě nutnosti průhledným).

Zkusili jsme načíst rastrový obrázek a do vzniklých polygonů vkreslit šestiúhelníky. Abychom se procvičili v R, zkusili jsme cyklus `for` přes barvy. Přes `data[]` jsme zjistili, že červená barva je v intenzitách 66, 55, 153 a 77. Příslušné RGB hodnoty jsme vyčetli v GIMPu. Zkušenější uživatelé R by jistě přišli na to, jak unikátní RGB hodnoty získat přímo v R a na tom postavit cyklus. Ručně to u 4 barev šlo, kdyby jich bylo víc, bylo by potřeba celý postup zautomatizovat.

Nepodařilo se nám hexa hodnotu barev vytáhnout z `data.frame` po spojení proměnných `hodnoty` a `barvy`. Zůstalo nám to jako otevřený problém.

Zde je výsledek našich snah.

```
library(raster); library(ggplot2); library(sf)
data<-raster("tux/ossconf-2020-upravene.png"); hodnoty<-c(66,55,153,77)


barvy<-c("#426baa","#37a9e9","#999999","#4d4d4d")
mojkovo<-ggplot()
for (volim in 1:length(hodnoty)) {
  polygony <- rasterToPolygons(data, dissolve=TRUE,
    fun=function(x){x==hodnoty[volim]})
  hexiky <- polygony %>% spsample(type = "hexagonal", cellsize = 8)
  nahled <- HexPoints2SpatialPolygons(hexiky, dx = 8)
  mojkovo <- mojkovo + geom_sf(data=st_as_sf(nahled), colour=barvy[volim],
    fill=NA)
  }
mojkovo <- mojkovo + theme_void(); mojkovo
#pdf("2020-logo-ossconf.pdf"); mojkovo; dev.off()
```



5. Druhý krok s knihovnou raster

Připravili jsme si obecný rastrový obrázek. Začali jsme v TEXu ve složce sazba sázet soubor pf2020.tex:

```
\documentclass{article}
\pagestyle{empty}
\usepackage{graphicx}
\begin{document}
\centering\bf\sffamily
\resizebox{9.5mm}{!}{PF}\par2020!\par ČStS%
\end{document}
```

Získali jsme postupně pdf a poté tif soubor v linuxovém prostředí:

```
$ lualatex pf2020.tex; \
> pdfcrop --hires --margins 5 pf2020.pdf; \
> gm convert -density 300 -monochrome pf2020-crop.pdf pf2020-crop.tif
```

V R jsme užili tento skript:

```
library(raster)
library(tidyverse)
library(sf)
data <- raster("sazba/pf2020-crop.tif")
polygony <- rasterToPolygons(data, dissolve=TRUE, fun=function(x){x>0})
hexiky <- polygony %>% spsample(type = "hexagonal", cellsize = 1.5)
as_tibble(hexiky)
nahled <- HexPoints2SpatialPolygons(hexiky, dx = 1.5)
ggplot() + geom_sf(data=st_as_sf(nahled), colour="brown", fill=NA)
```

Zajistili jsem si tak, že můžeme pracovat s libovolným černobílým rastrovým obrázkem a libovolným rozlišením. V této chvíli jsme však neužili skript hexwall, ale vyexportovali jsme si nalezené souřadnice středů šestiúhelníků.

```
write.table(hexiky@coords, "hexagony.csv", sep=",", col.names=FALSE)
```

První řádky souboru hexagony.csv vypadaly takto:

```
"1",62.7959799161181,21.6272033299488
"2",64.2959799161181,21.6272033299488
"3",65.7959799161181,21.6272033299488
```

6. Sesypání šestiúhelníkových nálepek

Objevili jsme dva velké repozitáře s nálepkami, stáhli jsem si je přes:

```
$ git clone https://github.com/rstudio/hex-stickers.git; \
> git clone https://github.com/Bioconductor/BiocStickers.git
```

Nakopírovali jsme si png do nové složky `pfhex`, u hex-stickers to bylo rychlé. U BioStickers jsme použili pomocný skript v Lua, který nám naparsoval `README.md` a vytáhl si png soubory nálepek z podadresářů:

```
soubor=io.open("README.md")
obsah=soubor:read("*all")
soubor:close()
kam=io.open("spust.sh","w")
unicode.utf8.gsub(obsah, "<img src=\"([^\"]-%.png)\"", function(s)
  print(s)
  kam:write("cp "..s.." ../hexwall-master/pfhex\n")
  end)
kam:close()
```

Soubor jsme spustili přes `texlua` dostupný v TeXLive a vzniklý dávkový soubor přes `sh`.

Posledním úkolem bylo připravit si podklady. Připravili jsme si složku `pfhex-output` a spustili následující (šikovný administrátor si snadno převede do skriptu):

```
$ cd pfhex; \
> for file in `find -type f -iname \*.png -printf "%f\n"`; do \
>   echo $file; \
>   gm convert $file -transparent white ../pfhex-output/$file; \
> done
```

Tím jsme zajistili, že jsou soubory průhledné, nezlobí tam barevný profil ICC (ImageMagick) a můžeme operativně zasáhnout do velikosti obrázků přes parametr `resize`.

7. Luujeme

Nyní máme stavební kameny a opustíme R. V TeXu je práce s datovými soubory možná, my použijeme Lua skript a v TeXu budeme jen sázet výsledek. Je to obdoba generování HTML přes PHP či JavaScript. Kvůli čitelnosti však nemícháme Lua skript uvnitř TeXu, což je možné, ale oddělíme jej.

Lua jako skriptovací jazyk se stal neocenitelným pomocníkem v TeXovém světě. Příchod LaTeXu3, nemluvě o ConTeXtu, sice umí mnohé, ale pro „obyčejné“ programátory je Lua jen jiná forma C++, Javy, JavaScriptu, Pythonu či Perlu. Ukažme si prvně střípky programování v Lua.

7.1. Výběr s vracením

Kdybychom chtěli vybrat 13 čísel od 1 do 52, můžeme to učinit takto:

```
for k=1,13 do
  print(math.random(1,52))
end -- končí cyklus for
```

Takový soubor bychom spustili přes `texlua` z TEXLive. Nabízí se možnost programu `lua`, např. z balíku `lua5.2`, případně `lua5.3` z balíku `lua5.3`.

7.2. Výběr bez vracení

Vytvoříme si prázdné pole a vyplníme jej hodnotami 1 až 52. Postupně volíme pořadí z pole a vypisujeme hodnotu na dané pozici. Tuto hodnotu pak z pole odebereme.

```
hodnoty={}
for x=1,52 do table.insert(hodnoty,x) end
for k=1,13 do
  vyber=math.random(1,#hodnoty)
  print(hodnoty[vyber])
  table.remove(hodnoty,vyber)
end -- končí cyklus for
```

7.3. Výběr bez vracení s vracením

Taková vánoční drobnost. U novoročenky jsme chtěli výběr bez vracení, ale nalezených šestiúhelníků jsme měli víc než nálepek. Místo nějaké formy stratifikovaného výběru jsme z pole hodnoty odebírali a jakmile bylo pole prázdné, vyplnili jsme si jej všemi dostupnými nálepkami znovu. Tím jsme zajistili, že jsou výběry náhodné, ale že se žádná nálepka neopakuje výrazně vícekrát než jiné. Základem Lua je práce s tabulkami, při jejich kopírování přebíráme jednotlivé položky (angl. deep copy), prosté „rovná se“ by nám nepomohlo.

Pojďme na školní ukázku. Z 52 čísel jich vybereme 117. Znak `#` nám zjistí aktuální velikost pole.

```
hodnoty={}; vsechny={}
for x=1,52 do
  table.insert(hodnoty,x)
  table.insert(vsechny,x)
end -- končí cyklus for
for k=1,117 do
  vyber=math.random(1,#hodnoty)
  io.write(hodnoty[vyber].." ") -- místo print
  table.remove(hodnoty,vyber)
  if #hodnoty==0 then -- deep copy
    for k,v in pairs(vsechny) do hodnoty[k]=v end
  end -- končí podmínka if
end -- končí cyklus for
```

Náš pokus by mohl dopadnout takto: 34 13 35 46 7 12 48 43 [...] 16 5 28.

Pozorný čtenář jistě brzy zjistí, že 39 hodnot se opakuje přesně dvakrát, 13 hodnot přesně třikrát. Obdoba by byla, když rozdáváme balíček karet, a jakmile jsme všechny karty rozdali, použijeme další balíček karet. Kdybychom rozdávali po 13

kartách, rozdali jsme karty 9 hráčům, spotřebovali bychom dva celé a jednu čtvrtinu balíčku žolíkových karet bez žolíků.

8. R + Lua + TEX

Bokem si uložíme z adresáře `pfhex-output` seznam nálepek:

```
$ ls *.png >../soubory-pfhex.txt
```

V R jsme provedli výpočty a získali jsme soubor `hexagony.csv`. Nyní si přes Lua skript tyto dva soubory načteme. Náš cíl je vygenerovat Ti*k*Zový zdrojový kód, který vysázíme. Navíc jsme si v Lua skriptu nastavili jednoduché měřítko. Výsledek našich snah by mohl vypadat takto:

```
math.randomseed(1)
soubory=io.open("soubory-pfhex.txt") -- seznam nálepek
obsah=soubory:read("*all")
soubory:close()
pngs={}
pngsfull={}

unicode.utf8.gsub(obsah, "([^\n]-)\n", function(s)
  table.insert(pngs,s) -- pracovní tabulka s nálepkami
  table.insert(pngsfull,s) -- neměnná tabulka všech nálepek
  end)
soubor=io.open("hexagony.csv") -- seznam nalezených souřadnic šestiúhelníků
obsah=soubor:read("*all")
soubor:close()
kam=io.open("zdrojak.tex","w") -- TikZový zdrojový soubor
kam:write("\\begin{tikzpicture}[remember picture, overlay]\n")
unicode.utf8.gsub(obsah, ",([^,]-),([^\n]-)\n", function(s,t)
  tos=s/1; tot=t/1 -- jednoduchá škála, je-li nutná
  pickup=math.random(1,#pngs) -- výběr bez vracení
  kam:write("  \\node[m] at ("..tos.."pt,"..tot.."pt)
    {\\includegraphics[width=\\maldimen]{"..pngs[pickup].."}};\n")
  table.remove(pngs,pickup) -- odebere vybrané logo
  if #pngs==0 then -- vrátit všechny nálepky, deep copy
    for k,v in pairs(pngsfull) do pngs[k] = v end -- for
  end -- if
end) -- unicode.utf8.gsub
kam:write("\\end{tikzpicture}\n")
kam:close()
```

Postupně se generuje soubor `zdrojak.tex`, první řádky vypadají takto:

```
\begin{tikzpicture}[remember picture, overlay]
  \node[m] at (62.795979916118pt,21.627203329949pt)
    {\includegraphics[width=\maldimen]{glue.png}};
```

```
  \node[m] at (64.295979916118pt,21.627203329949pt)
    {\includegraphics[width=\maldimen]{scmap.png}};
```

Náš poslední úkol je načíst si tento Ti*k*Zový kód a vysázet novoročenku, soubor `pf.tex`. To provedeme v TEXu. My jej měli ve složce `sazba`, aby se nám pomocné TEXové soubory nemíchaly s ostatními.

```
\documentclass[landscape]{article}
\pagestyle{empty}
\usepackage{tikz}
\graphicspath{ {../pfhex-output/} }
\newdimen\maldimen \maldimen=1.5pt % cellsize / škála v Lua souboru
\tikzset{inner sep=0pt, outer sep=0pt,
  m/.style={xshift=-100pt, yshift=-100pt, draw=none} }

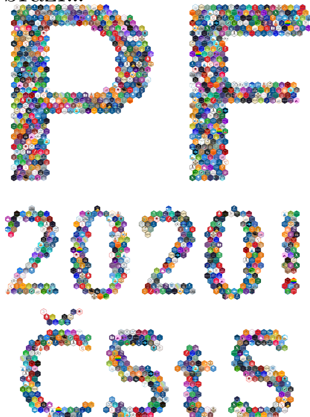

\begin{document}
\input ../zdrojak.tex
\end{document}
```

Nezbývá než si vše vysázet:

```
$ lualatex pf.tex; \
> lualatex pf.tex; \
> pdfcrop --hires --margins 0 pf.pdf; \
> gm convert -density 1800 pf-crop.pdf pf-crop.png
```

První dva řádky zajistí vysázení a absolutní umístění na straně. Třetí řádek nám ořeže ochrannou bílou zónu. Šikovný TEXista brzy zjistí, že by se to dalo zjednodušit na jeden běh TEXu bez nástroje `pdfcrop`. Necháváme otevřené pro badatele. Poslední řádek nám vygeneruje rastrový náhled.

Nezbývá než se pokochat vánočním dárkem, a doufat, že soubory pdf a png nebudou moc velké, nebude se to mezi svátky dlouho vykreslovat a do Nového roku se novoročenka celá zobrazí...



9. Závěrečné tipy: ještě jedno ohlédnutí za R

S určitým časovým odstupem si odpovídáme na otevřené otázky u R.

Při instalaci knihovny `rgdal` to chce novější verzi R než nabízí standardní linuxový repozitář (Xubuntu 18.04). Vyřešili jsme to takto.

V `/etc/apt/sources.list` jsme přidali:

```
deb https://cloud.r-project.org/bin/linux/ubuntu bionic-cran35/
```

Následovala instalace R:

```
$ sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys
    E298A3A825C0D65DFD57CBB651716619E084DAB9
$ sudo apt update
$ sudo apt upgrade
$ sudo apt install r-cran-base
```

V případě problémů se závislostmi užíváme místo známých programů `apt` či `apt-get` program `aptitude`. Hodilo se nám to při míchání 32 a 64bitových aplikací, resp. programů z různých linuxových distribucí.

V R následuje doinstalování knihoven, které použijeme. Ideální řešení je instalovat jednu knihovnu za druhou a sledovat případné chybové zprávy.

```
install.packages(c("png","spex","raster","rgdal","sf","pacman"))
```

9.1. Zjištění barvy konkrétního pixelu

Poněvadž jsme zápasili s datovými typy, zkusili jsme si získat barvu konkrétního pixelu a dostat jeho RGB složky, ve stylu GIMPu, ale už bez GIMPu.

Využili jsme knihovnu `raster` a funkci `brick` nebo `stack`, získali jsme složky RGB, převedli na šestnáctkové hodnoty a přidali znak `#`.

```
library(raster)
data<-brick("tux/ossconf-2020-upravene.png")  # nebo
#data<-stack("tux/ossconf-2020-upravene.png")
danaBarva<-paste("#", paste( as.hexmode( getValues(data,50,1)[50,] ), sep="",
    collapse=""), sep="")
danaBarva
```

Výstup je:

```
[1] "#426baa"
```

To už lze přímo použít pro parametr `colour`, např. `colour=danaBarva`.

9.2. Výpis všech barev v obrázku

Máme za sebou barvu jednoho pixelu, podívejme se, jak lze proces zautomatizovat a získat všechny jedinečné barvy v rastrovém RGB obrázku. Použijeme knihovnu `png`.

```
library(png)
mujpng<-readPNG("tux/ossconf-2020-upravene.png")
unique(as.raster(mujpng[,,1:3]))
```

Výstup je:

```
[1] "#FFFFFF" "#4D4D4D" "#426BAA" "#37A9E9" "#FEFEFE" "#FEFFFF" "#999999"
```

U našeho obrázku linuxového tučňáka Tuxe v logu konference OSSConf jsme zjistili, že máme dvě barvy (#426BAA, #37A9E9), dvě šedé (#4D4D4D, #999999), bílé pozadí (#FFFFFF) a na první pohled neviditelné, řekněme „parazitní“, chceme-li přehlédnuté, téměř bílé pixely: šedé a barevné (#FEFEFE, #FEFFFF).

Nyní už lze barvy oddělovat (například chceme mít co barva to jeden polygon), filtrovat (nechceme např. bílé pozadí ani padesát odstínů šedi), měnit (nahradit jednu barvu za jinou), spojovat (např. spojit barvy textů, nebo vše spojit do jedné barvy) ap.

9.3. Pracovní zobrazení obrázku

Zkusíme si jednoduchý filtr: odstranit bílé a téměř bílé pixely a zobrazit si pracovní náhled obrázku. Zároveň pro načtení více R knihoven otestujeme knihovnu pacman. Zmíněné otevřené problémy v R jsou tímto uzavřené.

```
#library(raster); library(sf); library(spex); library(ggplot2)
library(pacman) # alternativní postup načtení více knihoven
pacman::p_load("raster","sf","spex","ggplot2")
data<-raster("tux/ossconf-2020-upravene.png")
data[data>200] <- NA # ořez bílých a téměř bílých pixelů
vysledek<-polygonize(data)
#plot(vysledek) # je to pomalé, ale použitelné
#pdf("pracovni-nahled.pdf") # uložení pro bulletinek
ggplot()+geom_sf(data=st_as_sf(vysledek))
#dev.off() # uzavření ukládání pdf
```



Kontaktní adresa

Ing. Pavel Stříž, Ph.D., U Škol 940, Bučovice, okres Vyškov, 685 01, Česká republika,
E-mailová adresa: pavel@striz.cz